Iterativna lokalna pretraga

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Aleksa Voštić, Lazar Perišić, Anđela Križan, Anđela Janošević
akile9v@gmail.com, lakiwow95@gmail.com,
endzikri@gmail.com, andjelaj197@gmail.com

16. april 2020.

Sažetak

Osnovna ideja ove metaheuristike je da se pretraga fokusira na samo deo mogućih rešenja koja dobija od ugrađene heuristike. Iako jednostavna zbog svoje modularnosti, pokazuje odlične rezultate u praksi. U radu je prikazana njena implementacija, kao i pristup kojim se došlo do nje. Takođe, prikazane su neke od primena ovog algoritma.

Sadržaj

1 Uvod

Važnost algoritama visokih performansi za rešavanje teških optimizacionih problema ne može se potceniti, i u mnogim slučajevima jedine dostupne metode su metaheuristike. Prilikom dizajniranja metaheuristike, poželjno je da bude jednostavna, i konceptualno i u praksi. Prirodno, takođe mora biti efikasna i, ako je moguće, opšte namene. Ako metaheuristiku posmatramo kao jednostavnu konstrukciju za usmeravanje (specifične za problem) heuristike, idealan slučaj je kada se metaheuristika može koristiti bez ikakvog znanja o zavisnosti od problema.

Kako su metaheuristike postale sve sofisticiranije, ovaj idealan slučaj je gurnut u stranu u potrazi za većim performansama. Kao posledica toga, znanje specifično za problem mora biti inkorporirano u metaheuristiku da bi se dostiglo vrhunsko stanje. Nažalost, ovo čini granicu između heuristike i metaheuristike nejasnom, i mi rizikujemo da izgubimo i jednostavnost i opštost. Da bi se suprotstavili tome, približavamo se modularnosti i pokušavamo da dekompozitujemo metaheuristički algoritam na nekoliko delova, svaki sa svojom specifičnošću. Konkretno, želeli bismo imati potpuno opšti namenski deo, dok bi svako znanje specifično za problem ugrađeno u metaheuristiku bilo odvojeno u drugi deo. Konačno, u najvećoj mogućoj meri, radije ostavljamo netaknutu ugrađenu heuristiku (koju treba "voditi") zbog svoje potencijalne složenosti. **Iterativna lokalna pretraga** pruža jednostavan način da se zadovolje svi ovi zahtevi. Suština iterativne lokalne pretrage je da se izbegne zaglavljivanje u lokalnom minimumu tako što u više iteracija primenjuje lokalnu pretragu na novo generisano početno rešenje.

Svrha ovog rada je da se prikaže detaljan opis iterativne lokalne pretrage. Do sada je, uprkos svojoj konceptualnoj jednostavnosi, dovela do brojnih vrhunskih rezultata bez korišćenja previše znanja specifičnog za problem.

2 Ideja iza iterativne lokalne pretrage

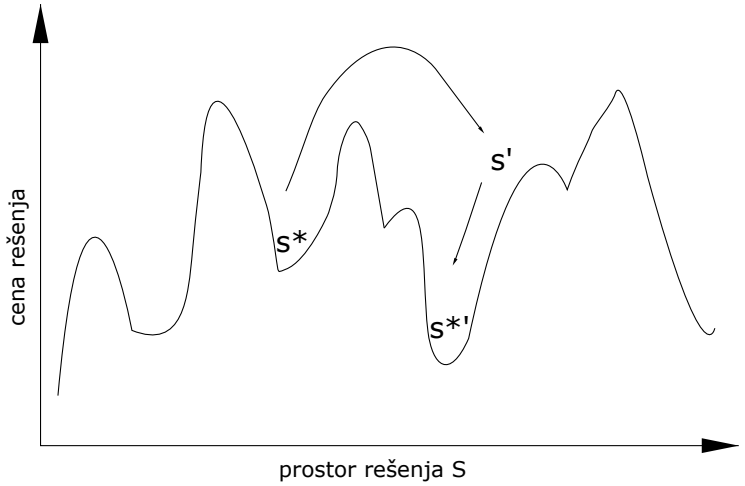
Pretpostavimo da imamo algoritam za specifičan problem koji aproksimira optimalno rešenje u okolini trenutnog rešenja. Taj algoritam ćemo zvati lokalna pretraga, iako to ne mora da bude *prava* lokalna pretraga i izvršavaćemo ga pozivanjem procedure `LokalnaPretraga`. Pitanje koje se postavlja jeste "Da li ovaj algoritam može da se poboljša korišćenjem iteracije?". Odgovor je da može, a rezultati dobijeni u praksi pokazuju da je to poboljšanje u većini situacija značajno [5].[1]

Neka je $\mathcal{C}$ funkcija cene nekog problema optimizacije. Kao i u većini slučajeva kada se pominje funkcija cene, tako i ovde, cilj nam je da tu funkciju minimizujemo. Potencijalna rešenja ovog problema ćemo označiti sa s, a skup svih tih rešenja sa $\mathcal{S}$. Naša lokalna pretraga tj. procedura `LokalnaPretraga` definiše preslikavanje iz skupa $\mathcal{S}$ u $\mathcal{S}^*$ koji predstavlja skup lokalno optimalnih rešenja s^*.

Osnovna ideja ILS (*eng.* iterated local search) tj. iterativne lokalne pretrage jeste da izbegne mane pokretanja lokalne pretrage iz nasumičnih delova prostora pretrage, odnosno skupa $\mathcal{S}$. Ovako dobijena rešenja nisu međusobno zavisna i lokalnu pretragu treba pozivati na ovaj način samo

[1] U retkim slučajevima kada iterativni metod nije pogodan za dati algoritam poboljšanje će biti minimalno.

kada druga rešenja ne daju bolje rezultate.[2] Uz to, kako se broj rešenja povećava, verovatnoća da nađemo kvalitetno s^* opada, što u praksi znači da kada broj rešenja teži beskonačnosti, kvalitetno, odnosno rešenje koje je blizu optimuma (globalnog), je nemoguće naći [3]. ILS se vodi idejom da umesto da koristi sva rešenja odnosno ona iz skupa $\mathcal{S}$, koristi samo podskup njih, odnosno skup $\mathcal{S}^*$. Cilj je da se krećemo kroz skup $\mathcal{S}^*$ i tako dođemo do globalnog optimuma ili bar rešenja bliskog njemu. Dakle, u svakoj iteraciji imamo trenutno rešenje s^* na koje vršimo perturbaciju (*eng.* perturbation), odnosno pomeramo se iz njega u međustanje s'. Ovo međustanje pripada $\mathcal{S}$ i na njega primenjujemo `LokalnaPretraga` da bismo došli do rešenja $s^{*\prime}$ iz $\mathcal{S}^*$. Sada odlučujemo da li prihvatamo ovo rešenje i nastavljamo od njega potragu ili se vraćamo na prethodno s^* rešenje. Ovu funkciju obezbeđuje procedura `KriterijumPrihvatanja`. Iterativnu lokalnu pretragu definišemo na način prikazan u algoritmu 1 a grafički prikaz iste se vidi na slici 1.



Slika 1: Grafički prikaz ILS. Na trenutno rešenje s^* primenjujemo `Perturbacija`, dobijamo s' na koje primenjujemo `LokalnaPretraga` nakon čega dobijamo novo rešenje $s^{*\prime}$.

Algoritam 1 Iterativna lokalna pretraga

s_0 = `GenerišiPočetnoRešenje`()
s^* = `LokalnaPretraga`(s_0)
repeat
  s' = `Perturbacija`(s^*, *istorija*)
  $s^{*\prime}$ = `LokalnaPretraga`(s')
  s^* = `KriterijumPrihvatanja`(s^*, $s^{*\prime}$, *istorija*)
until NIJE ZADOVOLJEN USLOV ZAUSTAVLJANJA[3]
return NAJBOLJE REŠENJE

[2] Iako se ovo retko dešava, ovaj pristup se u tim situacijama preporučuje zbog dobiti u jednostavnosti

[3] USLOV ZAUSTAVLJANJA je obično dozvoljeno vreme izvršavanja programa (*eng.* runtime) ili broja iteracija

3 Implementacija iterativne lokalne pretrage

Kao što možemo da vidimo iz prethodnog, da bismo implementirali algoritam tj. iterativnu lokalnu pretragu, potrebno je da implementiramo četiri komponente: `GenerišiPočetnoRešenje`, `LokalnaPretraga`, `Perturbacija` i `KriterijumPrihvatanja`.

3.1 Početno rešenje

Početno rešenje može biti veoma važno, pogotovo ako nam je cilj što brži dolazak do kvalitetnih rešenja. Generisanje početnih rešenja se može izvesti na dva načina. Možemo da koristimo (i) metodu slučajnog izbora, ili (ii) metodu pohlepne heuristike. U opštem slučaju, nije moguće reći koji metod od ova dva je bolji. Ipak, za kraća izvršavanja ILS preporučuje se metoda pohlepne heuristike, dok prilikom dužeg izvršavanja ILS izbor početnog rešenja nije mnogo bitan [5, 2].

3.2 Perturbacija

Perturbacije, odnosno pomeranja su veoma važna komponenta ovog algoritma. Naime, one služe da se pobegne od lokalnih minimuma (optimalnih rešenja) koji su često znatno lošiji od globalnog minimuma. ILS to čini tako što primenjuje perturbaciju na trenutni lokalni minimum. Definisaćemo snagu odnosno intenzitet perturbacije kao broj komponenti rešenja koji je perturbacijom promenjen. Snaga perturbacije je osobina koja najviše utiče na ovu komponentu algoritma. Naime, ako je snaga previše velika onda će se algoritam ponašati kao prilikom pokretanja lokalne pretrage iz nasumičnih delova prostora pretrage, a ako je previše mala, lokalna pretraga će često poništiti perturbaciju i kao novo rešenje naći ono od koga je krenula. Samim tim malo novih rešenja će biti istraženo. Ova veličina varira od problema do problema, a takođe zavisi i od veličine problema. Ponašanje ILS u praksi pokazuje da ne postoji jedinstvena optimalna snaga perturbacije. Ona se može menjati tokom izvršavanja i tada koristimo adaptivne perturbacije. Jedan način da se one implementiraju jeste korišćenjem istorije pretrage. Drugi način jeste deterministički određivanjem tokom izvršavanja [3, 5].

3.3 Kriterijum prihvatanja

Nakon što dobijemo novo moguće rešenje $s^{*\prime}$, na osnovu procedure `KriterijumPrihvatanja` odlučujemo da li to rešenje prihvatamo kao novo trenutno rešenje. Možemo imati neki od dva ekstremna pristupa:

- Uvek prihvatamo novo rešenje
- Ako je novo rešenje $s^{*\prime}$ bolje od s^*, onda ga prihvatamo, inače ne

Mnogi pristupi između ova dva su svakako mogući i često korišćeni. `KriterijumPrihvatanja` ima jak uticaj na prirodu korišćenih rešenja. Naime, u kombinaciji sa `Perturbacija`, može da se koristi za kontrolu ravnoteže između intenzifikacije[4] (*eng.* intensification) i diverzifikacije[5] (*eng.*

[4] Prihvatanjem samo boljih rešenja, smanjićemo prostor pretrage i pretraživati rešenja u trenutnoj okolini

[5] Prihvatanjem svih rešenja, povećavamo prostor pretrage i imamo veliki broj rešenja, ali često pretražujemo okoline sa lošim rešenjima

diversification). Kao i kod perturbacije možemo koristiti istoriju pretrage za dinamičko određivanje kriterijuma prihvatanja [3, 5].

3.4 Lokalna pretraga

Do sada smo lokalnu pretragu koju koristi ILS koristili kao crnu kutiju (*eng.* black box). Znali smo šta ta procedura radi ali nismo ulazili u detalje implementacije. Ipak, često nam je implementacija lokalne pretrage poznata i to nam omogućava da dodatno optimizujemo ILS. Postoji mnogo algoritama koji mogu da se koriste kao lokalna pretraga. Može se pretpostaviti da što je bolja lokalna pretraga to je bolji ILS. To često jeste tačno, ali postoje situacije kada lošiji algoritam daje bolja rešenja. Jedna od tih situacije je kada imamo ograničeno vreme izvršavanja programa i tada je nekad bolje koristiti brži, u teoriji lošiji algoritam i pozvati ga mnogo više puta nego bolji i sporiji algoritam. Zavisno od razlike u potrebnom vremenu, možemo dozvoliti više vremena za izvršavanje i koristiti bolji algoritam. Ako razlika u brzini dva algoritma nije velika, onda je obično isplativije koristiti bolji. Za određene probleme, često se kao lokalna pretraga koristi metaheuristika poput tabu pretrage (*eng.* tabu search), simuliranog kaljenja (*eng.* simulated annealing) itd [3].

4 Primene iterativne lokalne pretrage

ILS algoritmi su uspešno primenjeni u raznim kombinatornim optimizacionim problemima. U određenim slučajevima ovi algoritmi dostižu veoma visoke performanse. U ovom poglavlju prikazaćemo neke najčešće probleme.

4.1 Problem trgovačkog putika

Za dat skup gradova i cena putovanja između svaka dva grada, koja je najjeftinija ruta koja obilazi svaki grad tačno jednom, i vraća se u početni grad? Ovo pitanje postavlja problem trgovačkog putika. Ovaj problem izražen u terminima teorije grafova glasi: Za dat težinski graf (čiji čvorovi predstavljaju gradove, grane puteve između njih, a težine cenu putovanja) naći Hamiltonov ciklus najmanje težine.

Problem trgovačkog putnika je verovatno najpoznatiji kombinatorni optimizacijski problem. U suštini, on služi za testiranje izrade novih ideja algoritama, a dobre performanse na problemu trgovačkog putnika se uzimaju kao dokaz vrednosti takvih ideja. Kao i mnoge metaheuristike, neki od prvih ILS algoritama su testirani na ovom problemu.

Baum je prvi to uradio, osmislivši metodu ponovljenog spusta. On je u svojim testovima koristio tehniku 2-opt[6] kao ugrađenu heuristiku, nasumične 3-promene[7] kao perturbacije, i smanjivao je dužinu obilaska (odatle je i ime ove metode). Njegovi rezultati nisu bili zadovoljavajući, delom jer je razmatrao neeuklidski Problem trgovačkog putnika koji je suštinski teži u praksi od Euklidskog.

LSMC (Large-step Markov chain) algoritam je doneo velika poboljšanja. Martin, Otto i Felten su u ovom algoritmu koristili simulirano

[6] Preuređivanje dve grane koje se međusobno seku tako da presek više ne postoji.

[7] Heuristika lokalne pretrage koja je dosta proučavana je pravilo k-promena, gde se razmatraju one rute do kojih se može doći zamenom najviše k grana. Eksperimenti se uglavnom rade za k = 2, 3 ili 4.

kaljenje kao optimizaciju iz koje je izvedeno ime algoritma. Oni su razmotrili primenu 3-opt lokalne pretrage i Lin-Kernighan heuristike[8] (LK) koji je najbolji algoritam lokalne pretrage za problem trgovačkog putika. Ali, ključni momenat njihovog rada je uvođenje duplog mosta za perturbaciju, što je bilo bitno za dalji rad na euklidskom Problemu trgovačkog putnika.

D.S.Jonhson je za ILS koristio LK kao lokalnu pretragu, i osmislio je naziv "iterated Lin-Kernighan" (ILK). Razlike u imlementaciji u odnosu na LSMC su:

- potezi između dva mosta su nasumični
- cene se poboljšavaju (prihvataju se samo bolje)

Od ovih prvih studija predložene su i druge varijante ILS-a, a Johnson i McGeoch su dali sažetak stanja 1997 godine. ILS za problem trgovačkog putnika sa najvišim performansama je trenutno lančani LK kodiran od strane Applegate, Bixby, Chivatal i Cook, koji su detaljno opisali implementaciju. Oni su vršili eksperimente na ovom kodu uzimajući u obzir druge module:

- inicijalni obilazak
- izbori implementacije LK heuristike
- tipovi perturbacija

Njihovi testovi su vršeni na velikim instancama do čak 25 miliona gradova. Za potez sa dvostrukim mostom, razmatrali su efekat forsiranja ivica da budu kratke i istraživali su nasumične poteze sa dva mosta. Oni su zaključili da se najbolje performanse postižu kada su potezi dvostukog mosta usmereni prema kratkim dužinama ivica. Međutim, jačinu usmeravanja ka kratkim ivicama treba prilagoditi raspoloživom vremenu računanja, tj. što je kraće vreme računanja, ivice bi trebalo da budu kraće. Pri testiranju uticaja inicijalnog obilaska, došli su do zaključka da najgore performanse postižu slučajni početni obilasci, dok su najbolje rezultate dali Kristofidesov algoritam[9] i pohlepna heuristika.

Osim ovih radova, predložena su još dva ILS algoritma za problem trgovačkog putnika. Thomas Stützle je ispitao ponašanje ILS algoritma za problem trgovačkog putnika, i zaključio da ILS algoritmi stagniraju tokom dugog perioda izvođenja, što je i očekivano pri vršenju intenzivnog pretraživanja. Da bi se izbeglo stagniranje, predloženo je ponovno pokretanje i poseban kriterijum prihvatanja zbog variranja pretrage. Cilj te strategije je da pretraga nastavi od pozicije koja je veća od određenog minimalnog rastojanja od trenutne pozicije. Rezultati pokazuju da je ova metoda veoma efikasna, čak i kada se koristi samo 3-opt lokalna pretraga.

Drugi ILS algoritam od 1997. godine su razvili Katayama i Marisha. Oni su uveli novi mehanizam perturbacije i nazvali su ga genetska transformacija. Mehanizam genetske transformacije koristi dva obilaska, jedan od njih predstavlja do sada najbolje rešenje, a drugi je obilazak koji je pronađen ranije u pretrazi. Eksperimenti sa iterativnim LK algoritmom koji koristi genetsku transformaciju umesto standardnog dvostrukog mosta pokazuju da je pristup vrlo efikasan.

[8] Uključuje zamenu parova skupa grana kako bi se napravila nova ruta. Između ostalog koristi 2-opt tehniku. U svakom koraku odlučuje koliko grana je potrebno preurediti.

[9] Algoritam služi za nalaženje približnih rešenja problema trgovačkog putnika. Primenjuje se na specijalne slučajeve kod kojih su rastojanja simetrična.

4.2 Problemi raspoređivanja

ILS se takođe može uspešno primeniti i na probleme raspoređivanja kao što su Single Machine Total Weighted Tardiness Problem, Flow shop problem, Job shop scheduling problem kao i mnogi drugi.

4.2.1 Single Machine Total Weighted Tardiness Problem

SMTWTP je jedan od fundamentalnih kombinatornih problema optimizacije. Naime, problem se sastoji od skupa nezavisnih poslova sa različitim vremenima obrade, težinama (cenama), kao i rokovima do kad poslovi moraju biti završeni predviđenim da se obrađuju na jednoj datoj mašini. Ideja je minimalizovati ukupno kašnjenje sa datim težinskim koeficijentima čija je uloga da kvantifikuju važnost svakog posla ili označe troškove kašnjenja izvršavanja posla na mašini, odnosno utvrditi optimalan redosled izvršavanja ovih poslova na jednoj mašini.

ILS algoritam za rešavanje ovog problema zasnovan je na osnovu dinamičke pretrage, odnosno koristi tehnike dinamičkog programiranja kako bi pronašao najbolji potez koji se sastoji od skupa nezavisnih poteza razmene. Svaki takav potez razmenjuje poslove na poziciji i i j ($i \neq j$). Dva poteza razmene su nezavisna ukoliko se ne preklapaju, odnosno ukoliko za dve razmene gde jedna uključuje pozicije i, j a druga k, l važi da je $\min\{i, j\} \geq \max\{k, l\}$ ili *vice versa*.

Perturbacija se sastoji od niza proizvoljnih poteza razmene. Takođe se koristi i dobro poznato svojstvo SMTWTP-a: postoji optimalno rešenje u kojem poslovi koji ne kasne su poređani u nerastućem poretku na osnovu rokova do kad moraju biti završeni, te se ovo svojstvo koristi na dva načina:

- diverzifikovanjem pretrage u koraku perturbacije
- smanjuje se vreme računanja

Kao kriterijum prihvatanja, uvodi se backtrack korak: nakon β iteracija u kojima je svako novo najbolje lokalno stanje prihvaćeno, algoritam se restartuje sa najboljim pronađenim rešenjem do tada, pa je backtrack korak posebni izbor zavisnosti prošlih stanja koja su ugrađena u `KriterijumPrihvatanja`. [3, 1]

4.2.2 Flow shop problem

U FSP-u imamo n poslova koji treba da se izvrše na m mašina u identičnom redosledu, gde je unapred određen niz operacija obrade na mašinama. Takođe, svaka mašina može da izvršava najviše jedan posao, kao i da svaki posao može biti izvršavan na najviše jednoj mašini. Važi uslov da drugi poslovi ne mogu prekidati poslove koji se izvršavaju na mašinama. Postavlja se pitanje: na koji način treba pronaći permutaciju takvu da minimalizuje vreme završetka poslednjeg posla, odnosno pronaći odgovarajući redosled poslova koji će da optimizuje pojedine unapred određene operacije obrade.

Algoritam je zasnovan na vrlo jednostavnoj implemetaciji lokalne pretrage. Početno stanje se konstruiše uz pomoć NEH heuristike[10] , dok se perturbacija generiše pravljenjem pomoću dva različita tipa poteza:

[10] Jedna od najboljih heuristika za određivanje početnih stanja, za razliku od nasumičnog izbora, ona prioritet daje poslu sa najvećom sumom vremena izvršavanja

- zamenom koja razmenjuje pozicije susednih poslova

$$\pi = (\pi(1), ..., \pi(i), \pi(i+1), ..., \pi(n)) \rightarrow \pi\prime = (\pi(1), ..., \pi(i+1), \pi(i), ..., \pi(n))$$

- razmena proizvoljnih poslova koji nemaju nikakva ogranicenja na susedne

$$\pi = (\pi(1), ..., \pi(i), ..., \pi(j), ..., \pi(n)) \rightarrow \pi\prime = (\pi(1), ..., \pi(j), ..., \pi(i), ..., \pi(n))$$

Naime, ustanovljeno je da ovakva perturbacija, sa samo nekoliko poteza može dovesti do poprilično dobrih rezultata.

Za kriterijum prihvatanja može se koristiti da se uvek bira permutacija koja je bolja i da se onda ona i zadrži, međutim postoji i kriterijum koji je dosta bolji, i malo unapređeniji od prvog navedenog. Naime, ukoliko je novo rešenje $s^{*\prime}$ bolje od s^*, onda ga prihvatamo, a inače će se $s^{*\prime}$ prihvatiti uz odgovarajuću verovatnoću koja će zavisiti od konstantnog parametra (parametar temperature).

ILS se takođe može koristiti i za rešavanje flow-shop problema korišćenjem nekoliko stanja u nizu. Za svako stanje bi postojalo, umesto samo jedne mašine, sada više grupa identičnih mašina. Njihova metaheuristika bi imala dve faze koje se ponavljaju iterativno. U prvoj fazi bi se operacije dodelile mašinama i početno stanje bi se konstruisalo. U drugoj fazi bi se koristio ILS kako bi se pronašao bolji raspored za svaku mašinu u svakom stanju modifikovanjem niza operacija svake od mašina. Proces bi se ponavljao sve dok se zadovoljavajuće rešenje ne dobije. [7, 3]

4.2.3 Job shop scheduling problem

Problem raspoređivanja poslova (JSSP) je popularni problem optimizacije u informatici i operativnom istraživanju. Osnovna verzija problema je data sa n poslova koji imaju različita vremena obrade i oni moraju da se rasporede na m mašina različitih snaga. Svaki posao se sastoji od niza instrukcija koje se moraju izvršiti po unapred određenom redosledu kao i da se svaka instrukcija mora izvršiti na odgovarajućoj mašini. Neophodno je pronaći odgovarajuću permutaciju, odnosno redosled poslova koji će minimizovati ukupno vreme obrade[11].

Jedan od načina pristupanja ovom problemu korišćenjem ILS je definisala H.R. Lourenço [4]. Naime, primenom ogromnog broja računarskih testova, tražio se način generisanja inicijalnih rešenja, nekoliko algoritama lokalne pretrage, korišćenje različitih perturbacija kao i tri kriterijuma prihvatanja, i ti testovi su se poredili. Uočeno je da incijalna rešenja imaju veoma mali uticaj na sam algoritam, dok su ostale komponente poprilično važne, i to posebno različiti načini formiranja koraka perturbacije. Njena šema formiranja perturbacije je zasnovana na definisanju jedno-mašinskog ili dvo-mašinskog potproblema fiksiranjem broja promenljivih u trenutnom rešenju i rešavanjem ovih potproblema nekom od heuristika ili nekim polinomijalnim algoritmima. Ovaj način će raditi dobro jer lokalna pretraga ne može poništiti perturbaciju, kao i to da se nakon perturbacije rešenja teže da budu poprilično dobra i takođe imaju 'nove' delove koji su optimizovani.

[11] Odnosi se na period od početka obrade, pa do trenutka kad su svi poslovi završili obrađivanje

Drugi način su izložili Balas i Vazacopoulos koji bi koristio promenljivu heuristiku pretrage u dubinu, koju su nazvali Guided Local Search (GLS). Ideja se zasniva na konceptu drveta susedstva, gde svaki čvor odgovara rešenju, dok su listovi formirani izvođenjem razmene na pojedinim kritičnim delovima. Razvili su ILS algoritam ugrađivanjem GLS-a unutar procedure shifting bottleneck (SB)[12] i zamenjivanjem ciklusa ponovne optimizacije SB procedure sa nekoliko ciklusa GLS procedure, pa je takva procedura nazvana SB-GLS1. Kasnije je nastala i SB-GLS2 procedura koja kada se ustanovi raspored za sve mašine, onda se iterativno uklanja jedna mašina i GLS se primenjuje na manjem skupu koji čine preostale mašine, a onda se ponovo primeni GLS procedura na početnom skupu koji sadrži sve mašine.

5 Efektivnost i efikasnost ILS algoritma

U narednoj tabeli ćemo demonstrirati rezultate poređenja različitih poznatih algoritama primenjenim na Flow shop problemu kao i samo ponašanja ILS-a u odnosu na ostale.
Svi algoritmi su implementirani pod istim uslovima i pokretani na istoj mašini: programirani su na jeziku Delphi 6.0, pokretani na Athlon XP 1600+ (1400MHz) korišćenjem 512 MBytes RAM memorije.

Koristićemo instance koje je ustanovio Taillard, odnosno različite kombinacije poslova (20, 50, 100, 200, 500) sa datim mašinama (5, 10, 20) za koje je ustanovljeno da su izuzetno teški problemi pa samim time su i dobri primeri merila algoritama. Parametar po kojem ćemo upoređivati algoritme je dat sledećom formulom:

$$\% \text{ Vremenskog uvećanja u odnosu na optimalno rešenje} = \frac{Alg_{rez} - Opt_{rez}}{Opt_{rez}} \cdot 100$$

gde je Alg_{rez} rezultat dobijen korišćenjem nekog od algoritama za odgovarajuću instancu, a Opt_{rez} je optimalno rešenje date instance. Ono što se odmah može primetiti jeste da ukoliko je ovaj procenat visok, to znači da se algoritam ponaša loše, jer za toliko odstupa od optimalnog rezultata.

[12] Heuristika koja minimalizuje problem uskog grla koji se javlja u job shop problemu usled toga što neki poslovi mogu da drže resurse (mašine), dok je ostalim poslovima neophodna ta mašina, pa moraju da čekaju i tako i nastaje usko grlo

Tabela 1: Prosečni porast vremena potrebnog za izvršavanje u odnosu na najbolje poznato rešenje izraženo u procentima i prosečno procesorsko vreme u sekundama (navedeno u zagradi) [6]

Problem	SAOP	SPIRIT	GAChen	GAMIT	ILS
20x5	1.39 (≤0.5)	5.22 (≤0.5)	3.82 (≤0.5)	4.21 (≤0.5)	0.24 (4.01)
20x10	2.66 (≤0.5)	5.86 (≤0.5)	4.89 (≤0.5)	5.40 (≤0.5)	0.77 (4.09)
20x20	2.31 (≤0.5)	4.58 (≤0.5)	4.17 (0.60)	4.53 (≤0.5)	0.85 (4.63)
50x5	0.69 (≤0.5)	2.03 (≤0.5)	2.09 (0.77)	3.11 (≤0.5)	0.12 (6.38)
50x10	4.25 (0.60)	5.88 (0.52)	6.60 (1.00)	8.38 (0.52)	2.01 (9.94)
50x20	5.13 (1.04)	7.21 (0.97)	8.03 (1.45)	10.65 (0.96)	3.29 (11.82)
100x5	0.40 (0.60)	1.06 (0.53)	1.32 (1.79)	5.41 (0.52)	0.11 (15.31)
100x10	1.88 (1.10)	5.07 (1.03)	3.75 (2.26)	12.05 (1.02)	0.66 (18.79)
100x20	5.21 (2.09)	10.15 (2.00)	7.94 (3.24)	18.24 (1.99)	3.17 (24.04)
200x10	1.56 (2.29)	9.03 (2.25)	2.70 (5.97)	7.52 (2.20)	0.49 (33.73)
200x20	4.83 (4.59)	16.17 (4.51)	7.07 (8.18)	15.35 (4.50)	2.74 (41.80)
500x20	3.40 (39.48)	13.57 (39.70)	4.61 (55.30)	12.17 (37.82)	1.29 (192.03)
Average	2.81 (4.42)	7.15 (4.38)	4.75 (6.77)	8.92 (4.21)	1.31 (30.55)

Algoritmi koji su korišćeni u ovom testiranju su Osman-ov i Potts-ov SA[13] algoritam (SAOP), Widmer-ova i Hertz-ova procedura SPIRIT[14], Chen-ov GA[15] algoritam (GAChen), Hibridni GA sa lokalnom pretragom (GAMIT) koji je implementirao Murata, kao i ILS koju je implementirao Stützle radi poređenja rada ovih algoritama na konkretnom problemu.

Ono što se može uvideti iz priložene Table 1, da je ILS najefikasniji i daje najbolje rezultate, međutim, u proseku zahteva i najviše procesorskog vremena (malo više od pola minuta) i ovaj primer je tako konstruisan da nije zadato vremensko ograničenje rada algoritama na računaru. Međutim, ukoliko uvedemo kriterijum zaustavljanja, odnosno ograničimo procesorsko vreme, i u tom slučaju će ILS izaći kao pobednik.

Zaključak je da, uz to što je poprilično efikasan, što možemo primetiti sa sprovedenih testova, on je takođe poprilično jednostavan za kodiranje, dosta jednostavniji od ostalih navedenih. Na to treba dodati i činjenicu da se ILS može unaprediti sa dosta moćnijom lokalnom pretragom.

6 Zaključak

ILS poseduje mnoge poželjne karakteristike metaheuristike: jednostavan je, lagan za implementaciju, robustan i veoma efikasan. Suštinska ideja ILS-a leži u fokusiranju pretraživanja ne na celokupnom prostoru

[13] (*eng.* Simulated annealing) - probabilistička tehnika za određivanje ekstremuma date funckije.

[14] (*eng.* Streaming Pattern Discovry in Multiple Timeseries) - procedura pronalaženja aktuelnih šablona u višestrukim vremenskim serijama koja inkrementalno pronalazi korelacije i skrivene promenljive u numeričkim tokovima podataka.

[15] (*eng.* Genetic algorithm) - metaheuristički algoritam globalne optimizacije ili pretrage koji koristi tehnike inspirisane biologijom, odnosno genetske pojmove kao što su selekcija, mutacija, ukrštanje, nasleđivanje itd.

rešenja, već na manjem potprostoru koji je definisan rešenjima koja su lokalno optimalna za datu optimizaciju. Uspeh ILS-a leži u pristrasnom uzorkovanju ovog skupa lokalnih optimuma. Koliko će se ovaj pristup pokazati efikasnim, uglavnom zavisi od izbora lokalne pretrage, perturbacija i kriterijuma prihvatanja. Interesantno je da čak i kada se koriste naivne implementacije ovih delova, ILS može učiniti i mnogo bolje od slučajnog ponovnog pokretanja. Ali sa daljim radom takvim da su različiti moduli dobro prilagođeni trenutnom problemu, ILS može često postati konkurentan ili čak vrhunski algoritam. Ova dihotomija je važna jer se može izvršiti optimizacija algoritma progresivno, i tako se ILS može zadržati na bilo kojem željenom nivou jednostavnosti. Ovo, plus modularna priroda iterativne lokalne pretrage vodi do kratkog vremena razvoja i daje ILS-u prednost u odnosu na složenije metaheuristike. Kao primer toga, podsetimo se da ILS u osnovi tretira ugrađenu heuristiku kao crnu kutiju; tada je unapređivanje ILS-a da bi se iskoristio novi, bolji algoritam lokalne pretrage skoro neposredan. Zbog svih ovih karakteristika verujemo da je ILS obećavajući i moćan algoritam za rešavanje stvarnih kompleksnih problema u industriji i uslugama, u oblastima od finansija do upravljanja proizvodnjom i logistikom.

Za kraj, imajmo na umu da iako je sav ovaj pregled dat u kontekstu rešavanja problema kombinatorne optimizacije, u stvarnosti se veći deo onoga što smo obuhvatili može direktno preneti na probleme kontinuirane optimizacije. Gledajući unapred prema budućim pravcima istraživanja, očekujemo da će se ILS primeniti na nove vrste problema. Neki izazovni primeri su: (i) problemi gde su ograničenja vrlo ozbiljna i zato većina metaheuristika ne uspeva; (ii) višekriterijumski (*eng.* multi-objective) problemi, približavanje stvarnim problemima; (iii) dinamički problemi ili problemi u realnom vremenu (*eng.* real-time) u kojima podaci problema variraju tokom procesa rešavanja. Istraživanje ovih problema je jedva počelo, ali trebalo bi da dovede do algoritama lokalnog pretraživanja visokih performansi.

Literatura

[1] Matthijs den Besten, Thomas Stützle, and Marco Dorigo. *Design of Iterated Local Search Algorithms*, chapter An Example Application to the Single Machine Total Weighted Tardiness Problem, pages 441–450. 2001.

[2] Aleksandar D. Đenić. Rešavanje diskretnih lokacijskih problema primenom metode promenljivih okolina. Available at http://147.91.66.17/bitstream/handle/123456789/4744/Aleksandar_Djenic_phd.pdf?sequence=1. Accessed: 20. 03. 2020.

[3] Michel Gendreau and Jean-Yves Potvin. *Handbook of metaheuristics*. Springer, 2010.

[4] Helena R. Lourenço. *Job-shop scheduling: Computational study of local search and large-step optimization methods*. 1995.

[5] Helena R. Lourenço, Olivier Martin, and Thomas Stützle. A beginner's introduction to iterated local search. Available at https://www.researchgate.net/profile/Helena_Lourenco/publication/228085587_A_beginner's_introduction_to_Iterated_Local_Search/links/00b7d51a3bacd452c9000000.pdf, 06 2001.

[6] Ruben Ruiz and Maroto Concepcion. *A Comprehensive Review and Evaluation of Permutation Flowshop Heuristics*. 01 2004.

[7] Thomas Stützle. Applying iterated local search to the permutation. 2000.